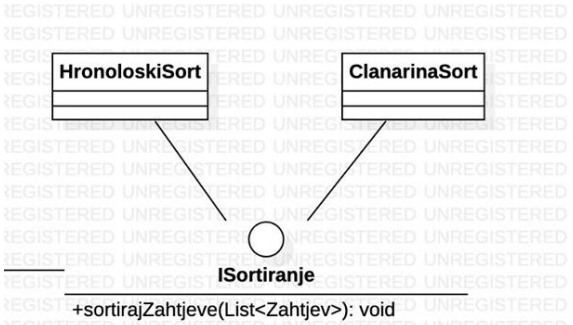
PATERNI PONAŠANJA

1. Strategy patern

Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji.

U našem sistemu smo implementirali ovaj patern tako što smo uveli sortiranje zahtjeva korisnika koje odobrava vlasnik. Dodali smo interface *ISortiranje*, i klase *HronoloskiSort* i *ClanarinaSort* koje predstavljaju sortiranje na osnovu zahtjeva korisnika po redu kako su pristizali, i sortiranje na osnovu tipa članarine.



2. State patern

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi.

Ovaj patern nismo implementirali u našem sistemu, ali ukoliko bismo to htjeli uraditi, mogli bismo dodati interface *IStatusClanarine*, i klase *AktivnaClanarina* i *NeaktivnaClanarina* koje bi implementirale taj interface, i metodu *platiClanarinu* kojom bi status članarine iz neaktivnog prelazio u aktivni.

3. Template Method patern

Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, no pojedinačne korake moguće je izvršiti na različite načine.

U našem sistemu nismo implementirali ovaj patern, ali mogli bismo to uraditi tako što uvedemo klasu *Sortiranje*, kao i klase *AbecedniSort*, *SlobodnaMjestaSort* i *PosjecenostSort*,

koje predstavljaju sortiranje parking lokacija prilikom prikaza po abecedi, po broju slobodnih mjesta i po posjećenosti parkinga.

4. Observer patern

Observer patern služi kako bi se na jednostavan način kreirao mehanizam pretplaćivanja. Pretplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama.

Ovaj patern nismo implementirali, ali mogli bismo to uraditi tako što bismo dodali mogućnost da korisnici koji imaju plaćenu čalanarinu primaju notifikacije o nekim promjenama vezanim za parking za koji su registrovani (npr. promjena kapaciteta, cijena i sl.). To bismo postigli tako što bismo dodali interface *IObserver* koji će imati metodu *posaljiObavjestenje*. U klasi EParking nam treba lista svih korisnika koji su platili članarinu, kao i metoda *obavijestiClanove*.

5. Iterator patern

Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste. Izvedba liste može biti u obliku drveta, jednostruke liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija.

Ovaj patern smo implementirali u našem sistemu, tako što smo omogućili prolazak kroz listu parking lokacija. Dodali smo interfejse *IKreatorIteratora* sa metodom *kreirajIterator*, interfejs *IIterator* sa metodom *dajSljedeciParking*, i klasu *ParkingIterator*, u kojoj se nalazi lista parking lokacija. U klasi *EParking* nam nije više potrebna lista parking lokacija, već dodajemo *parkingIterator*.